**Project1**

**Name: Shet Neha Nilcant and UTA ID: 1001387308**

The project consists of the multithreaded web server and the single threaded client where a web server serves a file on client request. The web server and the web client communicate using the Http. Since the server is multithreaded it can handle multiple requests from client at the same time. The server keeps listening for the request, on receiving a client request the server sets a TCP connection using the four tuples (Source IP, Source Port, destination IP, Destination Port). The server receives the data from client and checks for the requested file. If the file exists on the server, it will serve the client with the requested file with Http 200 OK code else it will send a 404 File Not Found error.

The server is set to listen on the default port number 8080, and the user can change the server port number when running the program through the command line. Below are the commands to start the server through command line:

1. python server.py

It will start the server and the server would listen on the port number 8080

1. python server.py <<port no>>

Here the port no would be any number between 0 to 65536. Eg: port number is 9001 then the command would be python server.py 9001. Now the server would listen for the request on the port number 9001.

Once the server is started and is waiting for the incoming request, we can check theWeb server implementation on the local machine using a Web browser. The url should also include the name of the file requested from the server. The file name could any file which is present on the server.

If the server contains check.html file.

**The url would be:**

<http://127.0.0.1:8080/check.html> **(In the first case)**

<http://127.0.0.1:9001/check.html> (**In the second case)**

If the user hits the wrong file name, the server will respond with HTTP/1.1 404 Not Found error.

Server code explanation:

* I have created a ServerThread which is a subclass of Thread and override \_init\_(self,args) method to pass the arguments which will be needed by the thread to perform the task when the thread is created. Then, override the run(self) which performs the actual task when the thread is started.

class ServerThread(Thread):

def \_\_init\_\_(self,ip,port,sock):

def run(self):

* The \_init\_() will initialize the ip,port and the socket object of the client for the new thread created. So, this client connection will be handled by the corresponding thread.

def \_\_init\_\_(self,ip,port,sock):

Thread.\_\_init\_\_(self)

self.ip = ip

self.port = port

self.sock = sock

print "A New thread has been created for Client Socket with IP:"+ip+" Port Number:"+str(port)

* Once the new thread instance is created you can start the thread using the start method which inturn calls the run() method. The run() method of the thread has all the logic to serve the client request.

def run(self):

try:

message = self.sock.recv(1024)

print "Data received from client \n"

print message+" \n "

filename = message.split()[1]

f = open(filename[1:])

outputdata = f.read()

f.close()

self.sock.send('HTTP/1.1 200 OK\nContent-Type: text/html\n\n')

#Send the content of the requested file to the client

for i in range(0, len(outputdata)):

self.sock.send(outputdata[i])

except IOError:

print 'exception code has been executed'

self.sock.send('HTTP/1.1 404 Not Found\nContent-Type: text/html\n\n')

self.sock.send('<p>HTTP/1.1 404 Not found: The requested file does not exist on this server.</p>');

self.sock.close()

The code is enclosed in try except to handle error generated in case of file reading. Next the self.sock.recv() will receive the incoming data from client and store it in the message variable. The message variable is operated accordingly to retrieve the requested filename. The file is opened and read and response header with 200 OK status is sent first and then the file content is sent to the server. If the file is not found on the server a 404 error is sent to the server. Once the request is served the socket connection is closed by the server.

* The socket function creates and return the socket object to use IP v4 and TCP.

serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

* The below option is used because the socket.SO\_REUSEADDR, causes the port to be released immediately after the socket is closed. Else once the program is restarted the port would be still in use and would give error.

serverSocket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

* The below block checks if the port number is specified as the command line argument otherwise it will take port number 8080 assigned to port variable.

if len(sys.argv) > 1:

port = int(sys.argv[1])

* Bind the socket to the ip address and the specified port name.

serverSocket.bind(("127.0.0.1", port))

* Server listens for the incoming connection

serverSocket.listen()

* Once the request comes, accept the request and obtain the connection object for the client.

Create a new thread and start the thread. As mentioned above it will insturn call the run method and serve the client request.

newThread = ServerThread(address[0],address[1],conn)

newThread.start()

**The Single Threaded Client:**

Run the client using the below format in command line

client.py < server\_IP address >< port\_no >< requested\_f ile\_name >

Eg: client.py 127.0.0.1 9001 check.html

If the request file is present on the server, it will respond with 200 OK and the file data will be displayed else a 404 file not found error will be displayed.

**Explanation for client code:**

Import the required modules sys,socket and time.

* Create a socket object to use IP v4 and TCP.

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

* Read the arguments from the command line which consist of ip name, port number and the filename and connect to the given server which is listening to the port number. Error handling is done if the user doesn’t specify all the arguments and will ask the user to enter all arguments.

if len(sys.argv) > 3:

ip = sys.argv[1];

port = int(sys.argv[2]);

s.connect((ip, port))

s.send("data /"+str(sys.argv[3]))

The data is sent to the client in “data/ <<filename>>” format. I have applied the logic here in this way so that I don’t need to change the server logic for receiving data and splitting differently for request through the browser or through this program.

* Once the request is sent to the client, I record the current time which would be my start time and similarly once the data is received from the server I would record the time again and that would be my end\_time. This is used for calculating the RTT value.
* receive the response from the server and When no more data is there break

data = s.recv(1024)

if not data:

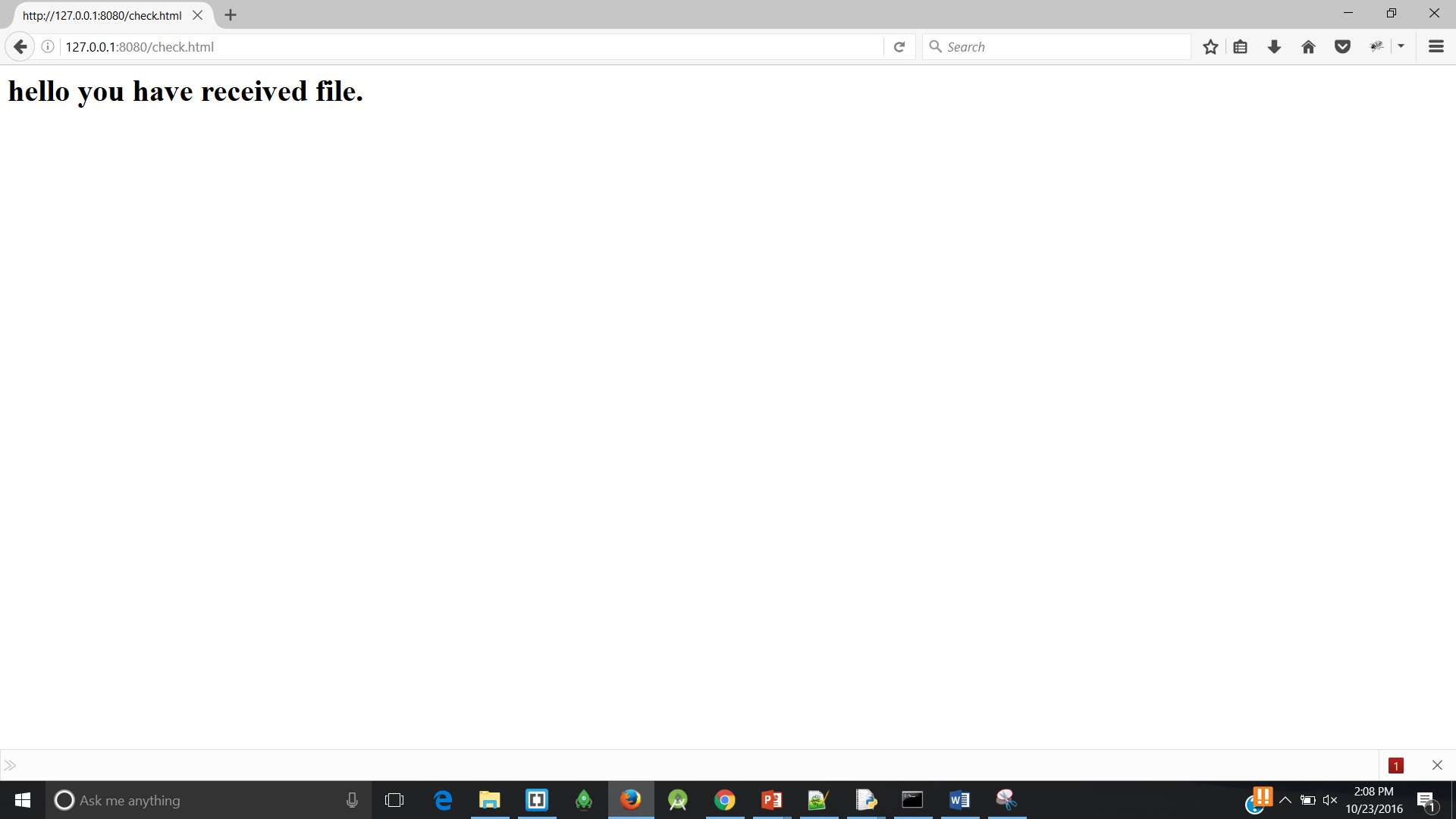
break;

* print all the necessary connection parameters and close the socket connect

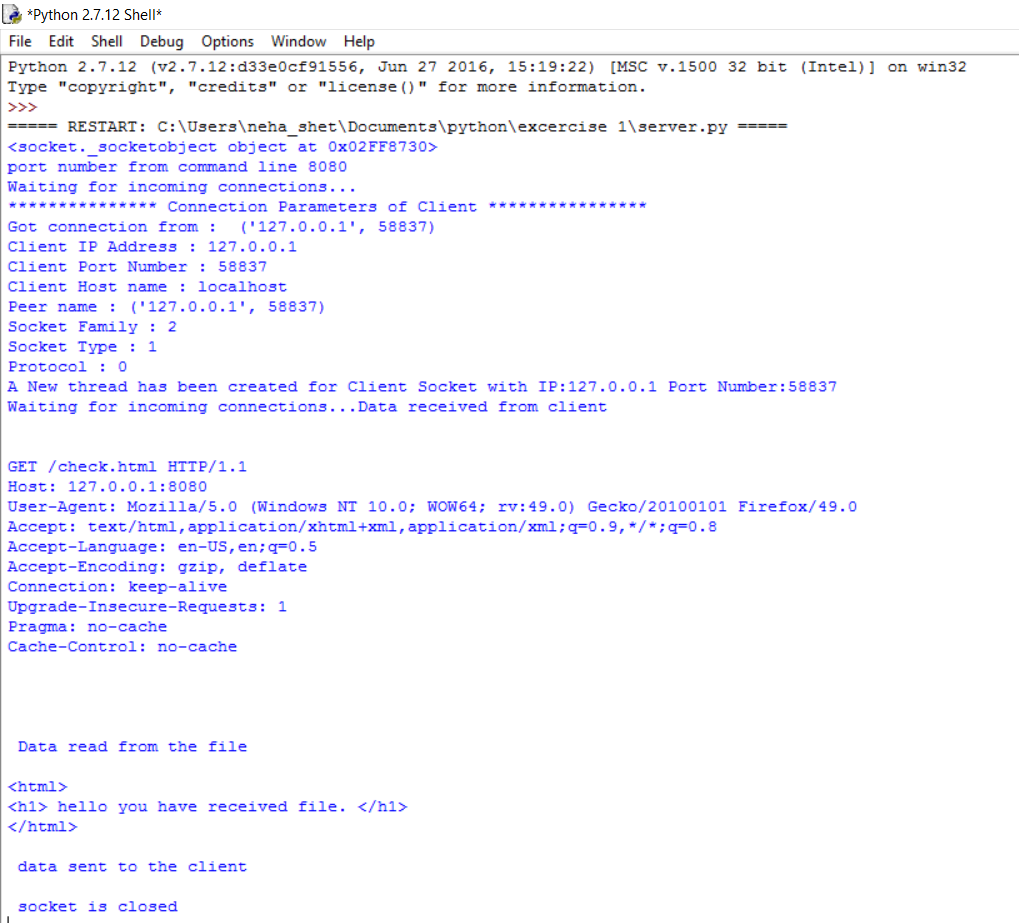
s.close();

Screenshots of the running server and Client:

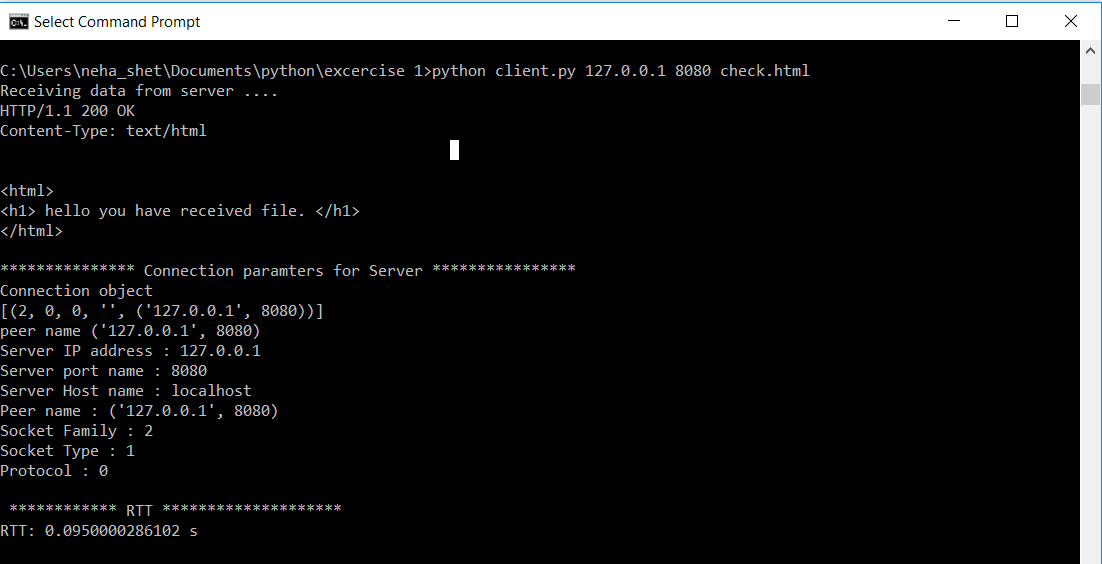
Test through browser:



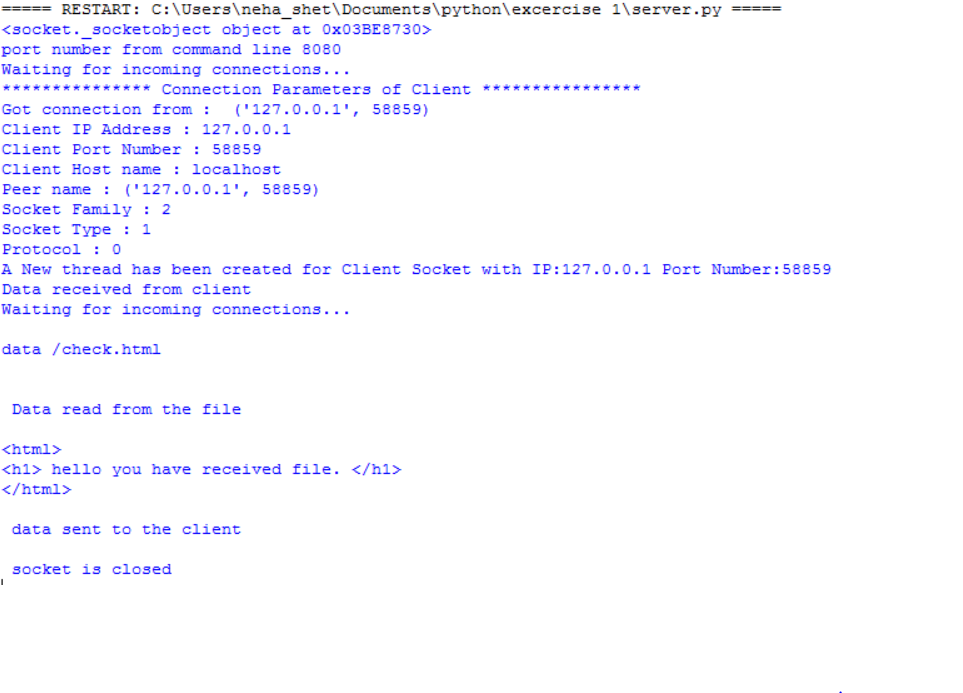
**Server logs:**



**Run the client.py program:**



**Server logs:**



**References:**

<https://docs.python.org/2/library/socket.html>

<https://www.tutorialspoint.com/python/>

<http://www.bogotobogo.com/python/python_network_programming_server_client_file_transfer.php>